An Evaluation of New and Old Similarity Ranking Algorithms

Paul Lynch, Xiaocheng Luan, Maureen Prettyman, Lee Mericle, Edward Borkmann, and Jonathan Schlaifer
U. S. National Library of Medicine, Lister Hill National Center for Biomedical Communications, Computer Science Branch
plynch@mail.nih.gov, luan@nlm.nih.gov, reenie@lhc.nlm.nih.gov

Abstract

The National Library of Medicine's (NLM) IRVIS project has been evaluating algorithms that determine which search results in a result set most closely match a selected search result. This is similar to the "find related" feature available on some websites (such as NLM's PubMed), but it operates within the context of search results that have already been retrieved. Essentially, it provides a way of re-ranking a results list in order of similarity to a particular result of interest to the user.

The similarity ranking algorithms that were tested included two set-based approaches as well as several variations on the traditional vector-cosine model. The algorithms were evaluated using the OHSUMED test collection from Oregon Health Sciences University. Surprisingly, the simpler set-based approaches provided a better ranking than the vector-based approaches and were also found to be faster.

1. Introduction

The NLM is developing a single access point to all of its online information collections, known as the Gateway (http://gateway.nlm.nih.gov). Search result sets in the Gateway can be large and difficult to navigate due to the heterogeneous data being accessed. Providing navigation assistance is one of the goals of the Information Retrieval and Visualization (IRVIS) project. The IRVIS project is exploring new ways of organizing and presenting search results, characterizing available data, and assisting the user in formulating queries. As a part of that work, we are designing a visualization that we call "Neighbor View", which is intended to show graphically which search results are most similar to a target search result selected by the user.

In order to develop such a system, we needed an algorithm for measuring the similarity of one search result's text to another. Some systems, such as the National Library of Medicine's (NLM) PubMed website (http://www.pubmed.gov), prepare special indexes in advance so that when a request is made for a document's "neighbors" the system only has to check static files [1]. In our case we did not have that option, because we are developing the visualization as a package which existing client systems (e.g. web-sites) could use. That design decision also meant that the Neighbor View package would not necessarily have access to the complete search result set all at once, but only to the sub-set given to it by the client system. It was therefore necessary to develop a similarity ranking algorithm that would perform its calculations on the fly and that would operate on the sub-set of search results given to it by the client system.

It quickly became clear that the code-base for the similarity ranking algorithms would be large enough to be a package in its own right, and we made the decision to separate it from the Neighbor View visualization. The Similarity Ranking Algorithms package now contains several ranking algorithms and a variety of term weight strategies. The following nine algorithms* are the subject of the testing being reported here.

2. Algorithms

Each of the rankers (ranking algorithms) take as input the ***target*** text string (the search result for which similar results are desired), and list of ***context*** strings (the sub-set of search results provided by the client system). The rankers work by comparing each context string against the target, getting a similarity score for each, and then ranking the results according to the scores.

Several of the rankers described below are based on the traditional vector space model introduced by Salton

* "Algorithm" here is used to denote a particular kind of similarity ranker in combination with some number of term weight strategies.

[2]. For these rankers, each string is regarded as having an associated vector of numbers* in which there is one number for each unique term in the entire context of strings, with the number representing that term's importance for that particular string. The index that corresponds to a given term is the same for each vector so that vectors for different strings can be compared. One can then consider the angle between two such vectors (in n-dimensional vector space, where n is the number of unique terms in the context) as a measure of the similarity between the texts. This is because an angle of zero would mean that the documents had identical term sets, while an angle of 90 would mean the documents did not share any terms. Usually the cosine of the angle is used as the measurement of similarity, which gives larger scores when the vectors are close together and smaller scores when they are farther apart. (For a more complete explanation of the vector-cosine approach, see Appendix II.)

The cosine of the angle can be computed by normalizing the two vectors and then taking the dot product. For the vector-based rankers, we implemented the normalization as one of several types of optional term weight strategies, which affect the "importance" of a term by changing the numbers stored in the vectors. This means that after applying any term weights, we can just use the dot-product for the score, i.e.:

$$S \equiv \sum_{t \in A \cap B} a_t b_t$$

where t is a term shared by both texts A and B, a_t is the term's weight in text string A's vector, and b_t is the term's weight in text string B's vector.

2.1. Set Ranker

This ranker computes a similarity score by treating the two strings as sets of (unique) terms **A** and **B**, and taking as the score **S** the ratio of the number of terms in the sets' intersection to the number terms in their union, i.e.:

$$S \equiv \frac{|A \cap B|}{|A \cup B|}$$

(This is the "Jaccard coefficient" for two sets.) The idea here is that if the strings are related to each other, the term sets will be more likely to have a larger intersection (and a smaller union). If the strings were identical, the intersection would equal the union, so dividing by the union's size gives us a number between 0 and 1.

2.2. Word Length

This ranker is very much like the Set Ranker, except that the terms are weighted according to their length, i.e.:

$$S \equiv \frac{\sum_{x \in A \cap B} length(x)}{\sum_{x \in A \cup B} length(x)}$$

This algorithm is based on the hypothesis that, in general, longer words are more likely to represent the subject of a text string than are shorter words. We were not sure this was correct, but it seemed plausible. (As discussed below, our tests indicate that this guess is correct; this algorithm outperforms the others listed here.)

2.3. Aslam-Frost

J. Aslam and M. Frost [3] have proposed an information-theoretic approach to measuring the similarity between strings of text based on work by D. Lin [4]. Their formula calculates the similarity score for two strings A and B as follows:

$$S = \frac{2 \cdot \sum_t \min\{p_{A,t}, p_{B,t}\} \log(\pi(t))}{\sum_t p_{A,t} \log(\pi(t)) + \sum_t p_{B,t} \log(\pi(t))}$$

where t is a term, π (t) is the fraction of context strings containing t, and $p_{A,t}$ is the 'fractional occurrence of term t' in A (the term count for t in A divided by A's word length).

2.4. Simple Vector

This is the simplest of the vector algorithms. The terms were completely unweighted, so the numbers in the vectors were either 1 (the term was present) or 0 (the term was absent).

2.5. Vector SW,N,IDF,PML

This algorithm is the same as the "Simple Vector" approach above, except that four types of term weights are applied (in sequence): Stop Words, Normalization, Inverse Document Frequency, and PubMed's local term weight. Each term weight strategy operates by multiplying the elements in a string's vector by a (possibly varying) number. Depending on the kind of term weight strategy, the factor that a given element in

* Although the concept of a vector was used, we found that there was a 50% time savings to do the implementation using hash maps (of terms to weights) instead of vectors.

the vector receives might depend on the term, the number of elements in the vector, or the context. These four strategies are described in detail below.

1. Stop Words (SW): Each element of the vector that corresponds to a term found in a set of 366 commonly occurring words is set to zero. The other weights are left at one.
2. Normalization (N): Each number in the string's vector is divided by the length of the vector (the square root of the vector's dot product with itself). This causes the vector to have length 1 (as measured in vector space).
3. Inverse Document Frequency (IDF): This is a commonly applied global weight (i.e. one that is based on the context as a whole rather than on the individual text string whose vector is being weighted). Each element in the vector is multiplied by a factor which is smaller if the term for that element occurs in many of the text strings in the context. Specifically, the weighting [5] used for a given term is:

$$\log_2\left(\frac{N}{n}\right)$$

   where N is the total number of strings in the context, and n is the number of strings containing the term. This formula gives terms that occur in fewer documents a higher weight than terms that are more common.
4. PubMed Local (PML): This is a local term weight (one that depends on the individual text string, rather than the context) used by the PubMed website. The formula [6] is:

$$l_w \equiv \frac{1}{1+e^{\alpha * d_{len}} * \lambda^{f-1}}$$

   where $\mathbf{l_w}$ is the local weight, α=0.0044, λ=0.7, $\mathbf{d_{len}}$ is the total number of terms in the string including duplicates, and **f** is the number of times the term occurs in the string. Because λ is less than 1, this formula gives a term a higher weight for a particular term when it appears frequently in that text string.

2.6. Vector SW,IDF,PML

This is the same as the previous algorithm except that the vectors are left unnormalized, to allow us to see the effects of normalization.

2.7. Vector SW,N,IDF,TF

This is the vector algorithm with four term weights applied: Stop Words, Normalization, Inverse Document Frequency, and Term Frequency (TF). The first three weights are the same as described above for "Vector SW,N,IDF,PML." Term Frequency is a typical local weight that gives terms a higher weight value in documents where they occur more frequently. The specific form of this weight we used was the augmented normalized term frequency [7], which is:

$$l_w \equiv 0.5 + 0.5 * \left(\frac{m_{dt}}{m_d}\right)$$

where $\mathbf{l_w}$ is the local weight for a term **t** in a string **d** (for "document"), $\mathbf{m_{dt}}$ is the number of times a term **t** occurs in **d**, and $\mathbf{m_d}$ is the maximum number of times any given term occurs in **d**.

2.8. Vector SW,IDF,TF

This is the same as the previous algorithm except that the vectors are left unnormalized, to allow us to see the effects of normalization.

2.9. Vector SW

This is the vector algorithm with only the Stop Words term weight applied.

3. Testing the Algorithms

We decided to test the similarity ranking algorithms using the OHSUMED test collection [8] (developed by Dr. William Hersh, and others) from Oregon Health Sciences University. The collection* consists of 5 files of 348,566 MEDLINE records (which are mostly journal article citations), a query file with 106 queries, and files which list ***documents*** (the MEDLINE records) that have been judged by people (physicians) to be either relevant, possibly relevant, or not relevant to the queries. The OHSUMED version of the records contains selected fields from the full MEDLINE record, including basic things such as title, author, and journal, as well as things like subject keywords (MeSH), the MEDLINE ID number, and the OHSUMED ID number. For most of the records in the collection, the abstract field is also present, but for about 1/3 no abstract is available. The abstracts that are present in the records are truncated, some at (roughly) 250 words and others at 400 words. (When truncated, they are

* The OHSUMED collection is available via anonymous ftp from medir.ohsu.edu.

marked as such.) In our use of the records, we used all of the available data, though we also tried testing without the abstract field for comparison (see below).

The queries in the OHSUMED query file are not necessarily the actual strings used to search the data, but are descriptions of the information that was being sought. For each query, several people tried different approaches to retrieve the desired information, and the documents retrieved by the query attempts were collected and judged for relevance. The total number of query-document pairs that were judged is 16,140. The other documents in the collection do not have associated judgments, but are less likely to be relevant to the queries because they were not retrieved.

The Similarity Ranking Algorithms rank text strings (e.g. the OHSUMED documents) according to their relevance to a target text string. The data provided with the OHSUMED collection does not directly indicate which documents are related to which other documents. However, it seems reasonable to make the assumption that the set of documents that have been judged relevant to a particular query are also relevant to one another. This might not be true in every case, but it seems likely to be true in most cases, and that should be sufficient to evaluate the ranking algorithms.

One thing the OHSUMED data does give us is the complete set of retrieved results (within the OHSUMED collection) from multiple attempts to search for documents for a given query. This was was very helpful, because it meant we did not need to write a search system for the OHSUMED data. Instead, we could use the queries in the query file, and take as their results the documents specified in the data.

For a given query, we obtained a list of the documents that were retrieved by it, and also information about the relevance judgments made for those documents. The relevance judgments were converted to a numeric scale, with 2 being definitely relevant, 1 being possibly relevant, and 0 being not relevant, so that each document had a query-relevance "score." (In cases where multiple judgments were made for a particular query-document pair, the scores were averaged.) If a query did not have at least two definitely relevant documents, it was skipped. The list of results was then given to one of the similarity ranking algorithms to rank, with the target text string being one of the definitely relevant documents.

To assess how well the ranking algorithm performed, a score was assigned based on how close to the top of the list the definitely relevant documents were placed. The ranked list was compared to a worst case list where the definitely relevant documents were all at the bottom of the list, and the definitely relevant documents were regarded (for the purposes of the scoring) as having moved from the worst case list position to the actual ranked list position. Each definitely relevant document was given one point for each notch moved in the right direction, and -1 point for each notch moved in the wrong direction. The sum of these points for the definitely relevant documents was taken as a measure of the ranking algorithm's performance.

In an ideal case, all of the definitely relevant documents would be moved to the top of the list. For a result set of size **n** with **d** definitely relevant documents, the sum **P** for this ideal case would be:

$$P = (n - d) * d$$

To get the normalized score for the ranking algorithm (so that the score would be independent of the result set size), we divided the actual sum of the document scores by the sum for the ideal case, which gave us a number between 0 (worst case) and 1 (ideal case). (For more detail on this scoring procedure, see Appendix III.)

That, then, is how the score for one ranking algorithm was computed for one query, using one definitely relevant document as the target text. We repeated the process for each definitely relevant document to get multiple scores for the algorithm for that one query, and tried using each algorithm and each query's result set (where possible), and were able to obtain 2,126 scores for each algorithm.

4. Results

Table 1 shows, as an example, the test results collected for the tenth query, for which there were five definitely relevant documents, each of which was used in turn as a target string for ranking the documents in the retrieval set (the target string document included). For space reasons, data is only shown for seven of the algorithms. At the bottom of each algorithm's column of scores are four rows of statistics that we computed for each query. For each algorithm, we computed the average of its scores (across the different target texts), the standard deviation, the number of times that algorithm outperformed all other algorithms being tested (the "Win Counts"), and the percentage of the time that the algorithm "won." (In the case of a tie, both algorithms were considered to win, which means that the percentages do not add to 100%.) The winning score for each trial is shown in bold.

4.1. Results for All Queries

The statistics for all 2,126 trials are shown in Table 2. The column and row headings have been switched in this table with respect to Table 1. These results are arranged in order from the highest average to the lowest.

Table 1: Sample Test Scores for OHSUMED Query #10

Target OHSUMED ID	Vector SW, IDF, PML	Set Ranker	Vector SW	Aslam-Frost	Vector SW, N, IDF, PML	Vector SW, N, IDF, TF	Word Length
6746	0.8267	0.66	0.7933	0.7867	**0.9133**	0.8067	0.6467
57590	0.8133	0.66	0.7467	0.7733	**0.9133**	0.78	0.64
111278	0.9067	0.6867	0.6733	0.82	**0.98**	0.8	0.7
310557	0.94	0.68	0.8	**0.9867**	0.96	0.9	0.6467
347443	0.9267	0.7	0.66	0.82	**0.9933**	0.92	0.72
Average	0.8827	0.6773	0.7347	0.8373	0.952	0.8413	0.6707
Std. Dev.	0.0586	0.0174	0.0656	0.086	0.0372	0.0638	0.0367
Win Counts	0	0	0	1	4	0	0
Win Percentages	0	0	0	20%	80%	0	0

Table 2: Summary of Results For All Queries

		Average Score	Std. Dev.	Wins	Win %
1	**Word Length**	0.6677	0.1106	904	43%
2	**Aslam-Frost**	0.6583	0.1067	466	22%
3	**Set Ranker**	0.6515	0.1100	223	10%
4	**Vector SW, N, IDF, PML**	0.6334	0.1064	151	7.1%
5	**Vector SW, IDF, PML**	0.6286	0.1037	30	1.4%
6	**Vector SW, N, IDF, TF**	0.6252	0.1028	38	1.8%
7	**Vector SW**	0.6246	0.1068	131	6.2%
8	**Vector SW, IDF, TF**	0.6168	0.1009	64	3.0%
9	**Simple Vector**	0.6065	0.1074	157	7.3%

Table 3: Summary of Results For All Queries, Without Using Abstracts

		Average Score	Std. Dev.	Wins	Win %
1	**Word Length**	0.69216	0.10742	1,109	52%
2	**Set Ranker**	0.67992	0.10596	425	20%
3	**Aslam-Frost**	0.65967	0.10544	326	15%
4	**Vector SW,N,IDF,TF**	0.61925	0.10215	53	2.5%
5	**Vector SW,N,IDF,PML**	0.61731	0.10207	46	2.2%
6	**Simple Vector**	0.61725	0.10542	65	3.1%
7	**Vector SW**	0.61717	0.10776	70	3.3%
8	**Vector SW,IDF,TF**	0.61536	0.10094	40	1.9%
9	**Vector SW,IDF,PML**	0.61385	0.10096	39	1.8%

4.2. The Effect of Abstracts

To test the effect of the documents' abstract fields on the performance of the algorithms, we also ran the tests using all of the OHSUMED fields except the abstract. The results of those tests are presented in Table 3.

5. Analysis of Results

5.1. Checking that the Distributions are Normal

In order to make sure that the standard deviations are meaningful, we created scatter plots of the distributions for several of the ranking algorithms' score sets. The result of one of those scatter plots is shown in Figure 1. The plot is using a bin size of 0.01 to group similar scores together.

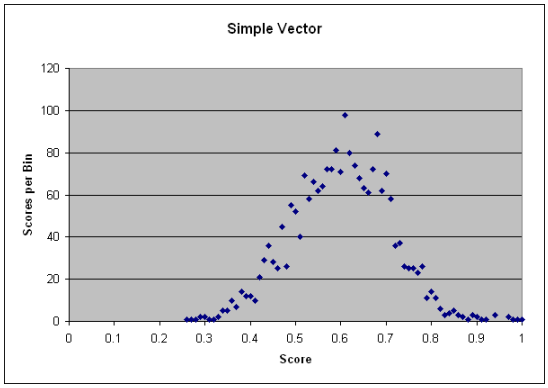


Figure 1: A plot of the distribution of scores obtained with the "simple vector" algorithm

All of the plots appeared to roughly follow the shape of a normal distribution, so calculating standard deviations was deemed appropriate.

5.2. Testing for Statistical Differences

The standard deviations for the algorithms' average scores produce ranges that all overlap each other, but that does not necessarily mean that the differences between the average scores are not significant. Using formulas and a table found in a handbook from NIST [9], a Perl script was written that would determine whether two averages were statistically different.* This test involves some chance of error, but the script was designed so that if it found two averages to be different, there would be a 95% confidence in that result. Using that test, we found that some of the averages really are better than the others.

The following tables summarize the results from the test script, using the data for each ranking algorithm. The column and row labels for tables 4 and 5 are the numbers assigned to the algorithms in tables 2 and 3, respectively. (Note that the ordering in tables 2 and 3 are slightly different, so the numbers are not the same between tables 4 and 5.) A "D" in a table box means the averages for the algorithms represented by the row and column are different (with 95% confidence.) A "-" in a box means the averages were not shown by the test to be different. Table 4 compares the average scores obtained from the tests that used abstracts (Table 2), while Table 5 compares the averages for the tests where abstracts were not used (Table 3).

Table 4: Differences Between Algorithms (Abstracts Included, c.f. Table 2)

	1	2	3	4	5	6	7	8
9	D	D	D	D	D	D	D	D
8	D	D	D	D	D	D	D	
7	D	D	D	D	-	-		
6	D	D	D	D	-			
5	D	D	D	-				
4	D	D	D					
3	D	D						
2	D							

Table 5: Differences Between Algorithms (Abstracts Excluded, c.f. Table 3)

	1	2	3	4	5	6	7	8
9	D	D	D	-	-	-	-	-
8	D	D	D	-	-	-	-	
7	D	D	D	-	-	-		
6	D	D	D	-	-			
5	D	D	D	-				
4	D	D	D					
3	D	D						
2	D							

The same type of analysis can be done to evaluate the difference in the algorithms' performance between the two test runs. The following table indicates the effect of removing the abstract as a field to be considered. In the second column, a value of "Improved" means that there was a statistically significant improvement in the scores

* Available on the web at: http://irvis.nlm.nih.gov/cgi/compareAvgs.pl. Contact the authors for the source code if desired.

as a result of leaving out the abstract. A value of "-" means the difference between the scores was not found to be significant, and a value of "Degraded" means that leaving out the abstract significantly hurt the algorithm's performance.

Table 6: Effect of Excluding the Abstract

Algorithm	Impact
Word Length	Improved
Aslam-Frost	-
Set Ranker	Improved
Vector SW,N,IDF,PML	Degraded
Vector SW,IDF,PML	Degraded
Vector SW,N,IDF,TF	-
Vector SW	Degraded
Vector SW,IDF,TF	-
Simple Vector	Improved

6. Conclusions

Several conclusions can be drawn from the analysis. First, the Word Length algorithm did significantly better, on average, than any of the other algorithms tested. This is interesting, because with the exception of the Set Ranker algorithm and perhaps the Simple Vector algorithm, the Word Length algorithm is simpler than the others. Simpler algorithms generally mean faster algorithms, and some basic time tests of our algorithms confirmed that, except for the Set Ranker, the Word Length ranker was faster than the others.[*] Both in terms of accuracy and speed, for this application, Word Length appears to be superior.

Second, removing the abstract significantly helped the two set-based algorithms (Set Ranker and Word Length), but generally hurt the vector-based algorithms. It is likely that the set-based algorithms are more sensitive to the "noise" terms in abstracts which are not representative of the subject, and that removing the abstract gives more weight to the title terms, which are usually carefully chosen to represent the topic of the article. The vector-based algorithms, on the other hand, have term weights which can handle the noise terms to some extent, and therefore some of them are able to benefit from the inclusion of the abstract. The notable exception to this is the Simple Vector algorithm, which did not use any term weights, and therefore reacted the same way to noise terms as did the set-based algorithms.

Third, the fact that the differences were small relative to the standard deviations, combined with the fact the best average score was under 0.67 (with abstracts included), suggests that perhaps the testing strategy employed had some weaknesses. It would have been nice if there had been a broader range in average scores. Also, given that the score produced by a random ranker (which we implemented) results in an average score of 0.5, we would have been happier to see scores of 0.8 or 0.9. (However, at the very least, we can be pleased that all the rankers did do much better than a random ranker.) Two assumptions that were made could have resulted in the lower scores:

1. **Documents judged definitely relevant to a particular query are relevant to each other.** This is certainly not true all of the time, and although we may have been correct in banking on its being true most of the time, this assumption would have resulted in the algorithms having imperfect data to process, which would have resulted in lower scores. An alternative approach was tried that modified this assumption (see Appendix I), but the results were quite similar.
2. **Documents with similar subjects share similar terms.** This is a common Information Retrieval assumption, because the only data you have for deducing a document's subject is its term set. As with the first assumption, the fact that this does not always hold true would result in lower scores. Some additional accuracy might have been attained if synonyms were considered, but we have not attempted that primarily because we anticipated that it would significantly slow down the algorithms.

Finally, it is interesting to note, for the Vector-based approaches, which term weights significantly improved the score. Three things stand out:

1. When abstracts are included, the use of stop words (Vector SW) resulted in a significantly better average than the Simple Vector algorithm. This is not too surprising, but it is good to see the confirmation.

[*] The ranking times of course depend on the efficiency of the code implementing the algorithms. However, the Simple Vector algorithm, which does not even perform normalization or stop word elimination, was still found to be slower than Word Length even after efficiency improvements had been made that cut its time in half. The improvement in speed of Word Length over Simple Vector is a small, but significant amount. However, once the extra term weights were added in to test the Vector SW,N,IDF,PML algorithm, the Word Length algorithm was found to be more than three times faster.

2. Tables 2 and 4 show that when abstracts are included, normalizing the vectors helps in at least some of the cases. Our significance test did not indicate a significant difference between algorithms 4 & 5 (Vector SW,N,IDF,PML, Vector SW,IDF,PML). However, that test was not designed to prove that there was no significant difference between the two, so it is possible that normalization helps even in that case.

3. The use of the PML (PubMed's local) term weight instead of TF resulted in significant improvements, as long as the abstracts were included in the data analyzed by the algorithms.

---

7. Appendix I: Alternate Evaluation Strategies

After considering the first possible weakness cited above in the conclusions, we decided to try a different approach in evaluating the performance of the ranking algorithms. We redesigned the evaluation software and divided the evaluation process into three pieces:

1. The assignment of numeric scores to query-document pairs, based on the relevance of the document to the query that retrieved it. The judgments were already made as a part of the OHSUMED data, so this step just required a conversion from non-numeric judgments to a numeric score.

2. The assignment of numeric scores to document-document pairs to indicate the relevance of the documents to each other. These scores were based on the query-document scores assigned in the first step. It was not necessary to do this for all document pairs, but only for the pairs where one document was a selected target for the ranking algorithms, and the second document was in the same result set for a given query.

3. The assignment of a score to an algorithm's ranking of the documents for a given query and target document, to measure how closely the algorithm's ranking matched the ideal ranking. The ideal ranking was defined as the ranking resulting from ordering the documents in a query's result set according to their document-document scores for the given target document.

7.1. The Second Strategy

In the first evaluation strategy (used in the main body of the paper), steps 1 and 2 were essentially the same. That is, for a given target document and query, a document's document-document score was the same as its query- document score. In the new strategy, the idea was to use additional judgment information from the OHSUMED data about the two documents. In particular, we made the document-document judgments independent of a particular query by combining the judgment data for any query in which both documents appeared in the result set.

In the second strategy, all query-document scores were between 0 and 1, and the product of two query-document scores taken from the same query was regarded as statement about the probability that those two documents were related to each other. In cases where there was more than one query **q** that retrieved both documents, the probability scores $\mathbf{p_q}$ for each query were combined into an overall probability **P** using the formula:

$$P = 1 - \prod_{q} (1 - p_q)$$

This formula results in a probability that is always greater than or equal to any of the individual probabilities. This probability **P** was taken as the document-document score. We adjusted the query-document score numbers to get document-document scores that were to our liking, and eventually settled on 0.9 for a document "definitely relevant" to the query, 0.6 for a document "possibly relevant" to the query, and 0.3 for a document that was not relevant.

Hopefully an example will make this a little clearer. Suppose you have two documents, whose OHSUMED ID numbers are 65403 and 12293, and that these documents appear in the result set for query 65 with query-document judgments "d" (definitely relevant) and "d", respectively. Each of these gets a query-document score of 0.9, and if that were the only query those two documents both appeared in, their document-document score would be 0.9*0.9 = 0.81.

Now, suppose you have those two documents appearing together in three queries, with the following query-document (Q-D) scores:

Query #	Q-D for 65403	Q-D for 12293
65	d (= 0.9)	d (= 0.9)
91	p (= 0.6)	n (= 0.3)
92	n (= 0.3)	n (= 0.3)

In this case, the three probabilities obtained for each query would be 0.81, 0.18, and 0.09, and these would be

combined using the formula above to get an overall probability of 0.86. (The exception to this method was when a document was being compared against itself for relevance. In that case, the document-document score was fixed at 1.0)

This strategy also required a change to the way a score was calculated for an algorithm's ranking of documents for a query. In this new approach, it becomes difficult to identify a sub-group of documents (such as the definitely relevant documents in the former strategy) which the ranker should move to the top of the list. Instead, we opted to consider the ranker's ordering of all of the documents in a query's result set.

An algorithm's ranked list of documents, with the most relevant document to the target first, was converted into a list of document-document scores. This list was then sorted to the ideal case (in which the document-document scores are arranged from highest to lowest) in a way that kept track of how the list's elements' indices had shifted in the list. The number of index positions each element had shifted was added to get a total number of points **D**. We also calculated the number of points $\mathbf{D_{max}}$ that would have resulted from going from the worst case ranking to the ideal case*, and then the ranking score was taken to be:

$$S = 1 - \frac{D}{D_{\max}}$$

7.1.1. The Second Strategy: Results

The algorithm scores obtained by this second evaluation strategy are shown below in table 7. The test runs that produced these numbers included the abstract field in the data given to the rankers. Because this analysis was done prior to our learning of the Aslam-Frost algorithm, that algorithm was not included in the testing.

As is evident on comparison with Table 2, this evaluation strategy did not yield the general improvement in the averages we were hoping for. In fact, in all cases it slightly lowered the scores and increased the standard deviations. On the positive side, it is good to see that that this strategy produced an identical ordering of the rankers as did the first strategy. (The algorithms, arranged according to their average score, show up in the same order in Tables 2 and 6.)

This increases our confidence in the accuracy of the results produced by the first strategy.

Table 7: Summary of Results For All Queries, Second Evaluation Strategy

		Average Score	Std. Dev.	Wins	Win %
1	**Word Length**	0.6384	0.1187	1,220	57
2	**Set Ranker**	0.6198	0.1178	272	13
3	**Vector SW,N,IDF,PML**	0.5977	0.1224	192	9.0
4	**Vector SW,IDF,PML**	0.5905	0.1199	54	2.5
5	**Vector SW,N,IDF,TF**	0.5887	0.1199	62	2.9
6	**Vector SW**	0.5883	0.1178	134	6.3
7	**Vector SW,IDF,TF**	0.5794	0.1177	78	3.7
8	**Simple Vector**	0.5702	0.1148	128	6.0

7.2. The Third Strategy

The third strategy we tried is the same as the second except for the final piece of the evaluation process, the assignment of a score to the algorithm's ranking. As in the second strategy, the ranked list of documents returned by the ranking algorithm was converted into a list of document-document scores, and compared to an ideal case ordering of those scores. In this case though, the comparison was made by summing the differences between corresponding elements of the two lists. More formally, let **A** be the list of document-document scores as obtained from the ranker, and let **B** be the ideal case list, defined as:

$$A = (a_0, a_1, ...a_n) \text{ and } B = (b_0, b_1, ...b_n)$$

Then we computed a number of points **D** for **A**'s distance from **B**, and a number of points $\mathbf{D_{max}}$ for the worst case ordering's distance from **B**, with the formulas:

$$D = \sum_{i=0}^{n} |a_i - b_i| \text{ and } D_{\max} = \sum_{i=0}^{n} |b_i - b_{n-i}|$$

From **D** and $\mathbf{D_{max}}$, the algorithm's score **S** is computed using the same formula as for the second evaluation strategy (see above).

* We were unable to prove that the number of points obtained in shifting the elements from a worst case ranking to an ideal case ranking results in the maximum number of points, as much as that seems to make sense. We were able to prove that swapping any pair of elements in the worst-case ordering will not result in a higher number of points.

7.2.1. The Third Strategy: Results

Unlike with the first two strategies, which gave a random ranker a score of roughly 0.5, the third strategy gave a random ranker a score of approximately 0.3. From that one might expect that the scores for the other algorithms would also be lowered by about 0.2, and that seems to be the case. Table 8 shows how the algorithms were scored by this third approach. Abstracts were included in the data given to the ranking algorithms.

Table 8: Summary of Results For All Queries, Third Evaluation Strategy

		Average Score	Std. Dev.	Wins	Win %
1	**Word Length**	0.4569	0.1027	979	46
2	**Set Ranker**	0.4407	0.1014	384	18
3	**Vector SW,N,IDF,PML**	0.4197	0.0939	220	10
4	**Vector SW,IDF,PML**	0.4130	0.0935	131	6.2
5	**Vector SW,N,IDF,TF**	0.41022	0.0916	96	4.5
6	**Vector SW**	0.4091	0.1048	227	11
7	**Vector SW,IDF,TF**	0.4005	0.0929	124	5.8
8	**Simple Vector**	0.3957	0.1013	167	7.9

Even if one adds 0.2 to these scores, they are still slightly below those obtained by the first strategy, though the standard deviations are slightly improved. However, once again we obtained the identical ordering of the algorithms, lending further support to the conclusion that the algorithms really do stack up against each other as listed. It should be noted though that although the scores appear in the same order, whether or not two algorithms' scores are statistically different can be affected by the choice of evaluation strategy. For example, while the scores listed in Table 2 for algorithms 3 and 4 were not shown to be statistically different (see Table 4), the scores for those same two algorithms in Table 8 (using the third evaluation strategy) are statistically different.

7.3. Alternate Evaluation Strategies: Conclusions

Neither of these alternate evaluation strategies were able to improve the overall scores. It may be that the fault for the lower numbers does not lie with the evaluation strategy, but in some other area. There are at least four other possible explanations:

1. **Faults with the data set:** Some of the query-document pairs were judged for relevance multiple times, and the judgments for a given pair were not always in agreement. Relevance judgments are subjective, and people make mistakes.
2. **Limitations of the data set:** The data set provides query-document judgments; any extrapolation from query-document judgments to document-document judgments will be rough.
3. **Limitations of the algorithms:** It may be that other similarity ranking algorithms not evaluated here would have performed better.
4. **Problems with subject matching:** As mentioned above, the assumption that documents with similar terms are about the same subject (and vice-versa) is not always true.

---

8. Appendix II: On the Vector-Based Model

This is a slightly more detailed description of the vector model than was provided in section 2. In the vector model, you consider the set of unique terms that exist in all of the text strings being compared. A particular text string is then represented as a vector, each of whose elements indicates whether a particular term occurs in that string. If the term is not present, the element is a 0; if the term is present, the element is (ignoring term weights for the moment) a 1.

For example, suppose our collection of text strings consists only of the two strings,

S1 = “lettuce tomato carrots spinach”
S2 = “tomato carrots onion onion”

There are five unique terms in the set. We chose an ordering for the terms, so that there will be a particular term associated with each index in the vectors. Let's pick the ordering:

(lettuce, tomato, carrots, onion, spinach)

Then the two strings S1 and S2 would have the corresponding vectors V1 and V2:

V1 = (1, 1, 1, 0, 1)
V2 = (0, 1, 1, 1, 0)

Although it is difficult (impossible might be a better word) to imagine a five-dimensional space in which these vectors can be plotted, one can easily imagine the two-dimensional plane that contains the two vectors,

and think two-dimensionally about the angle that lies between them. If the strings were identical, the vectors would be the same, and the angle between them would be zero. If the vectors did not share any terms, the angle between them would be 90 degrees. (To see that, consider a list of just two terms, and the case where the two vectors are (0, 1) and (1, 0)). It therefore makes sense to consider the angle, or the cosine of the angle, as a measure of how closely related the two strings are.

The cosine of the angle between two vectors of n elements can be calculated by taking the dot product and dividing by the lengths of the vectors:

$$\cos(angle) = \frac{V1 \cdot V2}{|V1||V2|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

where the a_i are the elements of V1 and the b_i are the elements of V2.

Often some sort of "term weight" will be used with the vector model. A term weight would have the effect of scaling the elements of the vectors in some way, to give certain terms more weight (making them have a bigger impact). For example, suppose we applied a term frequency weight that replaced the 1's in the vectors with a count of the number of times the term occurred in the corresponding string. In our example, the only duplicate word is “onion” is S2, so V1 would stay the same but V2 would become

V2 = (0, 1, 1, 2, 0)

This changes the length of V2 and its relative angle with V1. In this case, it moves V2 further away from V1 because "onion" is now emphasized and V1 doesn't have it.

---

9. Appendix III: On Scoring An Algorithm's Ranking

This section provides more detail on the procedure used for assigning a score to a ranking returned by one of the similarity algorithms. The procedure here is the one mentioned in the main body of the paper, as opposed to the two alternate strategies mentioned in Appendix I. The discussion that follows assumes that the explanation in the main body of the paper (under the section “3. Testing the Algorithms”) has already been read.

Suppose you have 5 documents, **u**, **v**, **x**, **y**, and **z**, which constitute the complete result set returned by a particular query. Also suppose that the query-relevance scores (based on the OHSUMED judgments) are: **u**=1, **v**=2, **x**=0, **y**=1, and **z**=2. Let us take **v** as the target document for the ranking, and suppose that a given ranker, upon examining the text of the documents, returned them in the following order (from most related to **v** to least related):

(x, v, u, z, y)

We now consider the list of query-relevance scores, **R**, that correspond to this ordering of the documents. This would be:

R = (0, 2, 1, 2, 1)

The ideal ordering,„**I**, would be :

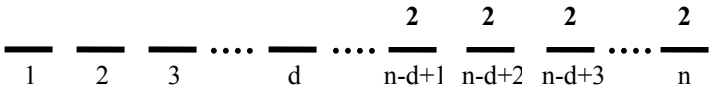
I = (2, 2, 1, 1, 0)

And the worst-case ordering, **W**, would be:

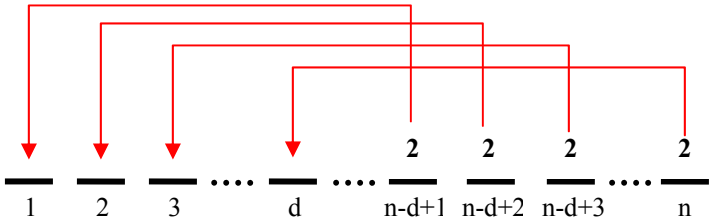
W = (0, 1, 1, 2, 2)

R is then regarded as having been created by shifting the 2’s (the definitely relevant documents) away from the **W** positions and toward the **I** positions, and the ordering in **R** is given a number of points equal to the number of places the 2’s have shifted. (For this evaluation strategy, we don't consider the other numbers.) Depending on how you map the 2's to each other, the number of points each 2 gets varies, but the total is the same. Picking one of the options, let’s say that the first two is one position off, and the second 2 is two positions off. This gives **R** a total of three (1+2) points.

So that we can have a score whose size is independent of the number of documents in a particular query's result set, we normalize the number of points by dividing by the maximum number of points. The maximum number of points occurs when **R** is in the ideal case scenario with all definitely relevant documents (2's) being on the far left. We can derive the general formula for the maximum points as follows. Suppose we have a worst case ordering of **n** elements with **d** definitely relevant scores at the far right, as shown below.



In reordering the list to the ideal case, each of the 2's gets shifted to the far left as shown below.



Each 2 is shifted by **n-d** places. Since there are **d** 2's, that gives us a total number of points **P** of:

$$P = (n-d)*d$$

In our example, we have **n**=5 and **d**=2, so **P**=6. This would mean that the ranking **R** gets a score of 3/6 = 0.5.

References

[1]http://www.ncbi.nlm.nih.gov/entrez/query/static/computation.html, 6/4/2003.

[2] G. Salton and M. E. Lesk. Computer Evaluation of Indexing and Text Processing. In *Journal of the ACM*, 15(1):8-36, 1968.

[3] J. A. Aslam and M. Frost. An Information-theoretic Measure for Document Similarity. In Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 449-450, 2003.

[4] D. Lin. An Information-Theoretic Definition of Similarity. In Proceedings of International Conference on Machine Learning, 1998.

[5] W. John Wilbur and Yiming Yang, An Analysis of Statistical Term Strength and its Use in the Indexing and Retrieval of Molecular Biology Texts. *Comput. Biol. Med.*, Vol. 26, No. 3, 1996, p. 210.

[6] Kim W, Aronson AR, and Wilbur WJ, Automatic MeSH Term Assignment and Quality Assessment. 2001 AMIA annual symposium proceedings, p. 319. (This formula was referenced by [1].)

[7] Wilbur & Yang, p. 211

[8] W. Hersh, C. Buckley, T. J. Leone, D. Hickam. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 192-201, 1994.

[9] NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/prc/section1/prc13.htm, 5/29/2003.